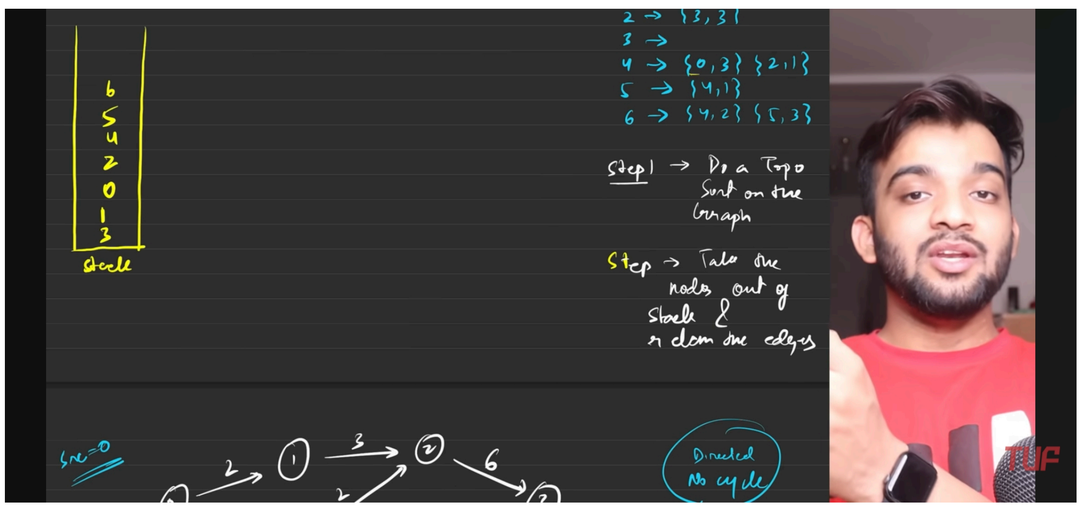

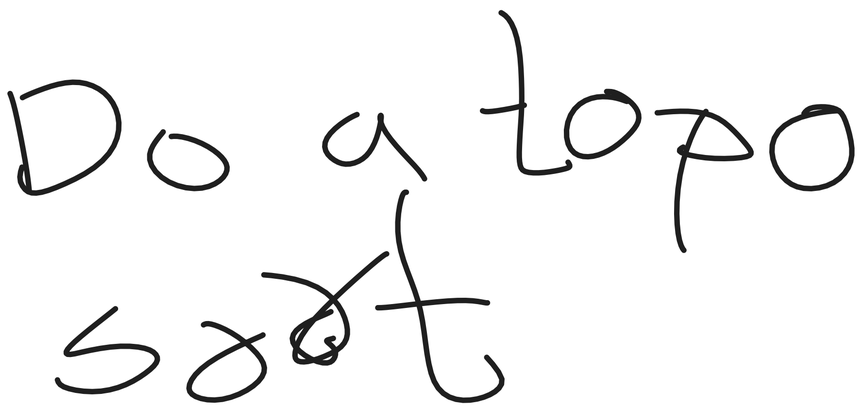

Relax the edges in this context simply mean we will check all the adjacent nodes and calculate the min distance to current neighbor/adjacent node using distance to current node plus the distance to neighbor from current and take min of existing distance to that node or new calculated distance.

I still need to add code snapshot

```
import java.util.*;
import java.lang.*;
import java.io.*;

class Main {

 public static void main(String[] args) throws IOException {
  int n = 6, m = 7;
  int[][] edge = {{0,1,2},{0,4,1},{4,5,4},{4,2,2},{1,2,3},{2,3,6},{5,3,1}};
  Solution obj = new Solution();
  int res[] = obj.shortestPath(n, m, edge);
  for (int i = 0; i < n; i++) {
   System.out.print(res[i] + " ");
  }
  System.out.println();
 }
}

class Pair {
 int first, second;
 Pair(int _first, int _second) {
  this.first = _first;
  this.second = _second;
 }
}
//User function Template for Java
class Solution {
 private void topoSort(int node, ArrayList < ArrayList < Pair >> adj,
  int vis[], Stack < Integer > st) {
  //This is the function to implement Topological sort.

  vis[node] = 1;
  for (int i = 0; i < adj.get(node).size(); i++) {
   int v = adj.get(node).get(i).first;
   if (vis[v] == 0) {
    topoSort(v, adj, vis, st);
   }
  }
  st.add(node);
 }
 public int[] shortestPath(int N, int M, int[][] edges) {
  ArrayList < ArrayList < Pair >> adj = new ArrayList < > ();
  for (int i = 0; i < N; i++) {
   ArrayList < Pair > temp = new ArrayList < Pair > ();
   adj.add(temp);
  }
//We create a graph first in the form of an adjacency list.

for (int i = 0; i < M; i++) {
 int u = edges[i][0];
 int v = edges[i][1];
 int wt = edges[i][2];
 adj.get(u).add(new Pair(v, wt));
}
int vis[] = new int[N];
//Now, we perform topo sort using DFS technique
//and store the result in the stack st.

Stack < Integer > st = new Stack < > ();
for (int i = 0; i < N; i++) {
 if (vis[i] == 0) {
  topoSort(i, adj, vis, st);
 }
}
//Further, we declare a vector 'dist' in which we update the value of the nodes'
//distance from the source vertex after relaxation of a particular node.
int dist[] = new int[N];
for (int i = 0; i < N; i++) {
 dist[i] = (int)(1e9);
}

dist[0] = 0;
while (!st.isEmpty()) {
 int node = st.peek();
 st.pop();
  for (int i = 0; i < adj.get(node).size(); i++) {
   int v = adj.get(node).get(i).first;
   int wt = adj.get(node).get(i).second;

   if (dist[node] + wt < dist[v]) {
    dist[v] = wt + dist[node];
   }
  }
 }

 for (int i = 0; i < N; i++) {
  if (dist[i] == 1e9) dist[i] = -1;
 }
 return dist;
}
}
```